I Erläuterungen

Voraussetzungen gemäß KCBG und Abiturerlassen BG jeweils in der für den Abiturjahrgang geltenden Fassung

Standardbezug
Die nachfolgend ausgewiesenen Kompetenzbereiche sind für die Bearbeitung der jeweiligen Aufgabe besonders bedeutsam. Darüber hinaus können weitere, hier nicht explizit ausgewiesene Kompetenzen für die Bearbeitung der Aufgabe nachrangig bedeutsam sein, zumal die Kompetenzen in engem Bezug zueinander stehen. Die Operationalisierung des Bezugs zu den Kompetenzbereichen des Standardbezugs erfolgt in Abschnitt II.

Aufgabe	Kompetenzen				
	K1	K2	K3	K4	K5
1.1		X		X	
1.2	X	X			
1.3		X		X	
1.4			X		
1.5			X		
1.6			X		
1.7.1	X				
1.7.2			X		
1.7.3		X		X	
1.7.4				X	X
2.1	X				
2.2		X			
2.3.1				X	
2.3.2				X	
2.3.3			X	X	
2.3.4			X	X	
2.3.5			X	X	
2.4		X	X		

Inhaltlicher Bezug
Die nachfolgend ausgewiesenen Themenfelder sind die wesentliche inhaltliche Grundlage für die vorliegenden Aufgaben. Darüber hinaus können weitere, hier nicht explizit ausgewiesene Themenfelder für die Bearbeitung nachrangig bedeutsam sein.
Q1: Objektorientierte Softwareentwicklung
Q2: Datenbanksysteme
Q3: Datenkommunikation
verbindliche Themenfelder: Objektorientierte Modellierung (Q1.1), Implementierung von Klassen und Assoziationen (Q1.2), Konzeptionelle und logische Modellierung einer Datenbank (Q2.1), Datenabfrage und Datenmanipulation mit SQL (Q2.2), Kommunikation in Rechnernetzen (Q3.2)

Nicht für den Prüfling bestimmt

II Lösungshinweise

In den nachfolgenden Lösungshinweisen sind alle wesentlichen Gesichtspunkte, die bei der Bearbeitung der einzelnen Aufgaben zu berücksichtigen sind, konkret genannt und diejenigen Lösungswege aufgezeigt, welche die Prüflinge erfahrungsgemäß einschlagen werden. Selbstverständlich sind jedoch Lösungswege, die von den vorgegebenen abweichen, aber als gleichwertig betrachtet werden können, ebenso zu akzeptieren.

<table>
<tr><th rowspan="2">Aufg.</th><th rowspan="2">erwartete Leistungen</th><th colspan="3">BE</th></tr>
<tr><th>I</th><th>II</th><th>III</th></tr>
<tr><td>1.1</td><td>beschreiben, erläutern
Das Klassendiagramm besteht aus den Klassen <code>Shop</code>, <code>Kunde</code>, <code>Warenkorb</code>, <code>Position</code>, <code>Bestellung</code> und <code>Produkt</code>. Eine Klasse beschreibt gleichartige Objekte mit ihren Eigenschaften (Attribute) und ihrem Verhalten (Methoden). Eine Klasse ist vergleichbar einem Bauplan für Objekte. Die zwischen den einzelnen Klassen bestehenden Assoziationen werden durch Linien zwischen den daran beteiligten Klassen dargestellt. Die Multiplizität bezeichnet die Wertigkeit einer Assoziation, d.h. sie spezifiziert die Anzahl der an der Assoziation beteiligten Objekte (hier: * für „0, 1 oder mehrere“ und 1 für „genau ein“).
Die Klasse <code>Shop</code> ist die Hauptklasse des Systems, sie kennt die Produkte, Kunden und Bestellungen über einseitig navigierbare Assoziationen. Das bedeutet, das Produkt-, Kunden- und Bestellungsobjekte das Shop-Objekt nicht kennen.
Die Beziehung zwischen den Klassen <code>Kunde</code> und <code>Warenkorb</code> ist ein Beispiel für eine beidseitig navigierbare Assoziation. In dieser Beziehung tritt der <code>Kunde</code> in der Rolle <code>kunde</code> auf, der <code>Warenkorb</code> in der Rolle <code>wk</code>. Ein anderes Beispiel für die beidseitig Navigierbarkeit ist die Beziehung zwischen den Klassen <code>Kunde</code> und <code>Bestellung</code>.
Eine Aggregation (nicht ausgefüllte Raute) beschreibt eine Assoziation zwischen einem Ganzen und seinen Teilen, ein Ganzes "besteht aus" seinen Teilen. Eine Aggregation besteht zwischen den Klassen <code>Warenkorb</code> und <code>Position</code>.
Zwischen den Klassen <code>Bestellung</code> und <code>Position</code> besteht eine Komposition (ausgefüllte Raute). Sie ist eine starke Form der Aggregation. Auch hier handelt es sich um eine Ganzes-Teil-Beziehung. Löscht man eine Bestellung, so werden auch alle Bestellpositionen gelöscht. Assoziationen können um Einschränkungen (constraints) ergänzt werden.
beschreiben
erläutern</td><td></td><td>

3
3</td><td></td></tr>
</table>

Aufg.	erwartete Leistungen	BE I	BE II	BE III
1.2	zeichnen (Objektdiagramm, siehe unten)	5		
1.3	implementieren (siehe unten)	2	4	3

1.2 zeichnen

kunde : Kunde
kdnr = 4711
name = Musterfrau
vorname = Erna
plz = 65185
ort = Wiesbaden
strassse = Kirschgasse 78
eMail = erna@musterfrau.de
passwort = geheim

: Position
menge = 3

wk : Warenkorb
summe = 47,55

: Position
menge = 2

p1 : Produkt
prodnr = 1000200
bezeichnung = Druckerpapier
preis = 3,95
lagermenge = 500

p2 : Produkt
prodnr = 1000300
bezeichnung = Druckertine schwarz
preis = 17,85
lagermenge = 20

1.3 implementieren

```
public class Kunde {
  private static int autowert=0;
  private int kdnr;
  private String name;
  private String vorname;
  private int plz;
  private String ort;
  private String strasse;
  private String eMail;
  private String passwort;
  private Warenkorb wk;
  private List<Bestellung> bestellungen;

  public Kunde(Kundendaten kdaten) {
    this.kdnr = ++autowert;
    initKundendaten(kdaten);
    this.passwort = generierePw();
    this.wk = new Warenkorb(this);
    this.bestellungen = new List<>();
  }

  public void leereWarenkorb() {
    wk.loescheAllePositionen();
  }
  public void hinzufuegenBestellung(Bestellung best) {
    bestellungen.add(best);
  }
}
```

Nicht für den Prüfling bestimmt

Aufg.	erwartete Leistungen	BE I	BE II	BE III
	(Code siehe unten)			

```
public class Warenkorb {
  private Kunde kunde;
  private List<Position> wkPositionen;

  public Warenkorb(Kunde kunde) {
    this.kunde = kunde;
    wkPositionen = new List<>();
  }

  public void hinzufuegenPosition(Produkt p, int menge) {
    boolean gefunden = false;
    for (Position pos : wkPositionen) {
      if (pos.getProdukt().getProdnr() == p.getProdnr()) {
        pos.setMenge(pos.getMenge() + menge);
        gefunden = true;
      }
    }
    if (!gefunden)
      wkPositionen.add(new Position(menge, p));
  }

  public double berechneSumme() {
    double summe = 0;
    for (Position pos : wkPositionen) {
      summe += pos.getMenge() * pos.getProdukt().getPreis();
    }
    return summe;
  }

  public void loeschePosition(int index) {
    if (index < wkPositionen.size()) {
      Position pos = wkPositionen.remove(index);
    }
  }

  public void loescheAllePositionen() {
    wkPositionen.clear();
  }

  public int getAnzahlPositionen() {
    return wkPositionen.size();
  }

  public Position getPosition(int index) {
    return wkPositionen.get(index);
  }
}
```

Aufg.	**erwartete Leistungen**	**BE I**	**BE II**	**BE III**
1.4	entwickeln, zeichnen shop : Shop wk : Warenkorb kunde : Kunde b : Bestellung bestellen(wk) pruefeVerfuegbarkeit(wk) {ergebnis} opt [ergebnis = true] getKunde() {kunde} <<create>> Bestellung(kunde) hinzufuegenBestellung(b) loop [für alle Objekte in wk] getPosition(aktuellerIndex) {p: Position} hinzufuegenPosition(p) loeschePosition(aktuellerIndex) updateLagermengen(b) {ergebnis}			
	entwickeln		3	3
	zeichnen	4		

<table>
<tr><th rowspan="2">Aufg.</th><th rowspan="2">erwartete Leistungen</th><th colspan="3">BE</th></tr>
<tr><th>I</th><th>II</th><th>III</th></tr>
<tr><td>1.5</td><td>modellieren, zeichnen

Shop
+ bestellen(wk : Warenkorb, za : Zahlungsart) : boolean

Vorkasse
– iban : String
– zahlungsFrist : int
+ Vorkasse(iban : String, frist : int)

– zahlungsarten 1..*

Zahlungsart
– id : int
– bezeichnung : String

PayPal
+ PAYPALADRESSE : String

– zahlungsart 1

Kreditkarte
– anbieter : String
+ Kreditkarte(anbieter : String)

Bestellung
+ Bestellung(kunde : Kunde, za : Zahlungsart)

Hinweis: Die Methode <code>bestellen()</code> der Klasse <code>Shop</code> und der Konstruktor von <code>Bestellung</code> sind zu erweitern.

modellieren
zeichnen</td><td>

1</td><td>

2</td><td>

3</td></tr>
</table>

Nicht für den Prüfling bestimmt

Aufg.	erwartete Leistungen	BE I	BE II	BE III
1.6	entwickeln, zeichnen **berechnePruefziffer(kartennr: String)** stellen := Länge von kartennr summe := 0 zähle i von 0 bis stellen-2 Schrittweite 1 zahl := kartennr an der Stelle i i mod 2 = 0? J: faktor := 2 N: faktor := 1 quer := quersumme(zahl*faktor) summe := summe + quer mod10 := summe mod 10 pruefziffer := 10 - mod10 return pruefziffer			
	entwickeln		3	2
	zeichnen	3		
1.7.1	erklären Threads sind leichtgewichtige Prozesse, die sich den gleichen Adressraum im Speicher teilen. Innerhalb eines Prozesses kann es mehrere Threads geben. Threads werden in Softwaresystemen dafür genutzt, um parallele Abläufe zu erzeugen.		2	
	beschreiben Der Server bekommt über den Konstruktor die Referenz auf den Onlineshop übergeben und hat anschließend über die Variable `shop` Zugriff auf den Onlineshop. Er eröffnet mithilfe der übergebenen Portnummer einen Server-Socket und wartet dann auf eingehende Client-Verbindungen. Nimmt ein Client Kontakt mit dem Server auf, wird dem Server eine Socketinstanz zurückgegeben. Der Server erzeugt nun eine neue Instanz der Klasse `UserThread`, die Zugriff auf die Socketinstanz und den Onlineshop hat. Ab diesem Zeitpunkt läuft die gesamte Kommunikation zwischen Client und dem Onlineshop über die erzeugte Instanz der Klasse `UserThread`. Die Methode `run()` in der Threadklasse implementiert den Kommunikationsablauf.	2		

Nicht für den Prüfling bestimmt

Aufg.	erwartete Leistungen	BE I	BE II	BE III
1.7.2	entwickeln		2	2
1.7.3	implementieren		3	3

1.7.2 entwickeln

Shop

Server
- serverSocket : ServerSocket
+ Server(port : int, shop : Shop)
+ start()
+ bestimmeAnzahlAktiverThreads() : int
+ abmeldenUserThread(ut : UserThread)
- wartung() : boolean

UserThread
- client : Socket
- server : Server
+ UserThread(client : Socket, shop : Shop, s : Server)
+ run()

AnzeigeThread
+ AnzeigeThread(s : Server)
+ run()

Thread

Server → Shop: - shop, 1; UserThread → Shop: - shop, 1; Server <<creates>> UserThread; Server <<creates>> AnzeigeThread; AnzeigeThread → Server: - server, 1; UserThread → Server: - server, 1; Server → UserThread: - userThreads, 0..*; UserThread, AnzeigeThread erben von Thread

1.7.3 implementieren

```
public class Server {
  private ServerSocket serverSocket;
  private Shop shop;
  private List<UserThread> userThreads;

  public Server(int port, Shop shop){
    this.serverSocket = new ServerSocket(port);
    this.shop = shop;
    this.userThreads = new List<>();
    new AnzeigeThread(this).start();
  }

  public void start(){
    while (true) {
      Socket client = serverSocket.accept();
      if (!wartung()) {
        UserThread t = new UserThread(client, shop, this);
        userThreads.add(t);
        t.start();
      } else {
        client.write("Shop nicht verfügbar\n");
        client.close();
      }
    }
  }

  public void abmeldenUserThread(UserThread ut) {
    userThreads.remove(ut);
  }

  public int bestimmeAnzahlAktiverThreads() {
    return userThreads.size();
  }
}
```

Aufg.	**erwartete Leistungen**	**BE**		
		I	**II**	**III**
1.7.4	erläutern Es kann passieren, dass ein Thread die Verfügbarkeit eines Produkts überprüft und signalisiert bekommt, dieses Produkt sei verfügbar. Anschließend wird der Thread unterbrochen und ein zweiter Thread überprüft die Verfügbarkeit desselben Produkts. Auch er bekommt signalisiert, dass das gewünschte Produkt verfügbar ist. Auf diese Weise könnten Produkte bestellt werden, die nicht in ausreichender Anzahl im Lager verfügbar sind.		2	
	Summe 60	**17**	**27**	**16**

Aufg.	**erwartete Leistungen**	**BE**		
		I	**II**	**III**
2.1	nennen, erläutern − Datenunabhängigkeit Die Unabhängigkeit wird durch die Trennung der physischen Speicherung der Daten und deren Verwaltung von den Anwendungsprogrammen erreicht. Ein Anwendungsprogramm muss nicht die Struktur der Daten kennen. − Sicherung der Integrität Durch die Anwendung der im konzeptionellen Schema vorgegebenen Integritätsbedingungen kann die logische Richtigkeit (Konsistenz) der Daten gewährleistet werden (in sich richtige und widerspruchsfreie Daten). − Synchronisation Ein Datenbankmanagementsystem (DBMS) hat die Aufgabe, die parallel ablaufenden Transaktionen (Folge von Lese- und Schreib-Operationen) aller Benutzer zu synchronisieren, d. h. die Zugriffe so zu verwalten, dass die Integrität der Daten erhalten bleibt. − Zugriffssteuerung Bei einem DBMS kann der Datenbankadministrator Zugriffsrechte für jeden Benutzer bzw. für Benutzergruppen vergeben. Hinweis: Das Erläutern anderer Vorteile ist zulässig.			
	nennen	2		
	erläutern		2	

2.2	überführen
	`kunde(kdnr,name, vorname, strasse, plz, ort, iban)`
	`bestellung(bestellnr, datum, kdnr#)`
	`rechnung(rechnnr, datum, bestellnr#)`
	`position(bestellnr#, posnr, menge, vkpreis, produktnr#)`
	`produkt(produktnr, bezeichnung, listenpreis, lagermenge)`
	`liefert(liefnr#, produktnr#,ekpreis)`
	`lieferant(liefnr, firmenname, ort)`
		7		

2.3.1	angeben
	`UPDATE produkt`
	`SET listenpreis = listenpreis * 1.05`
	`WHERE produktnr = 4713;`
		2		

2.3.2	formulieren
	`INSERT INTO bestellung values (4348, NOW(),`
	`(SELECT kdnr FROM kunde`
	`WHERE name = 'Wagner' AND vorname = 'Benjamin'));`
			3	

Nicht für den Prüfling bestimmt

Aufg.	erwartete Leistungen	BE I	BE II	BE III

2.3.3	entwickeln
	`SELECT p.produktnr, p.bezeichnung,`
	`       SUM(menge * vkpreis) AS Umsatz`
	`FROM bestellung b JOIN position po USING (bestellnr)`
	`     JOIN produkt p USING (produktnr)`
	`WHERE MONTH(b.datum) = MONTH(NOW())`
	`AND YEAR(b.datum) = YEAR(NOW())`
	`GROUP BY p.produktnr`
	`ORDER BY Umsatz DESC`
	`LIMIT 1;`
			2	3

2.3.4	entwickeln
	`SELECT DISTINCT p.produktnr, p.bezeichnung`
	`FROM position po JOIN produkt p USING (produktnr)`
	`WHERE po.bestellnr IN`
	`  (SELECT b.bestellnr`
	`   FROM bestellung b JOIN position po USING (bestellnr)`
	`   WHERE YEAR(b.datum) = YEAR(NOW())-1`
	`   AND po.produktnr = 4349)`
	`AND p.produktnr <> 4349;`
			2	2

2.3.5	entwickeln
	`SELECT p.produktnr, l.firmenname AS 'günstigster Lieferant'`
	`FROM lieferant l JOIN liefert li USING (liefnr)`
	`     JOIN produkt p USING (produktNr)`
	`WHERE lagermenge < 20`
	`AND li.ekpreis = (SELECT MIN(ekpreis)`
	`                  FROM liefert lief`
	`                  WHERE  lief.produktnr = p.produktnr)`
	`ORDER BY p.produktnr;`
			2	3

2.4	modellieren, zeichnen Rechnung [0,1] betrifft [0,n] zahlungswunsch identnr datumeingang sendet [0,n] Rücksendung [1,1] besitzt [1,n] Rücksendeposten id menge grund [1,1] Kunde [0,n] aufgegeben bei annahmedatum [1,1] Annahmestelle id bezeichnung [0,n] betrifft [0,1] Produkt modellieren zeichnen	 2	4	4
	Summe 40	**13**	**15**	**12**

III Bewertung und Beurteilung

Die Bewertung und Beurteilung erfolgt unter Beachtung der nachfolgenden Vorgaben nach § 33 der Oberstufen- und Abiturverordnung (OAVO) in der jeweils geltenden Fassung. Bei der Bewertung und Beurteilung der sprachlichen Richtigkeit in der deutschen Sprache sind die Bestimmungen des § 9 Abs. 12 Satz 3 OAVO in Verbindung mit Anlage 9b anzuwenden.

Bei der Bewertung und Beurteilung der Übersetzungsleistung in den Fächern Latein und Altgriechisch sind die Bestimmungen des § 9 Abs. 14 OAVO in Verbindung mit Anlage 9c anzuwenden.

Der Fehlerindex ist nach Anlage 9b zu § 9 Abs. 12 OAVO zu berechnen. Für die Ermittlung der Punkte nach Anlage 9a zu § 9 Abs. 12 OAVO sowie Anlage 9c zu § 9 Abs. 14 OAVO wird jeweils der ganzzahlige nicht gerundete Prozentsatz bzw. Fehlerindex zugrunde gelegt.

Für die Bewertung in den modernen Fremdsprachen ist der „Erlass zur Bewertung und Beurteilung von schriftlichen Arbeiten in allen Grund- und Leistungskursen der neu beginnenden und fortgeführten modernen Fremdsprachen in der gymnasialen Oberstufe, dem beruflichen Gymnasium, dem Abendgymnasium und dem Hessenkolleg“ vom 7. August 2020 (ABl. S. 519) zugrunde zu legen. Demnach erfolgt die Bewertung und Beurteilung mit der Maßgabe, dass lediglich bei der Ermittlung des Prüfungsergebnisses (Note) aus Prüfungsteil 1 und 2 gerundet wird.

Darüber hinaus sind die Vorgaben der Erlasse „Hinweise zur Vorbereitung auf die schriftlichen Abiturprüfungen (Abiturerlass)“ und „Durchführungsbestimmungen zum Landesabitur“ in der für den Abiturjahrgang geltenden Fassung zu beachten.

Als Kriterien für die Bewertung und Beurteilung dienen unter Beachtung der Zielsetzung der gymnasialen Oberstufe nach § 1 Abs. 2 OAVO neben dem Inhaltlichen auch die in den Kerncurricula genannten überfachlichen Kompetenzen, insbesondere die Sprachkompetenz und Wissenschaftspropädeutik; dies zeigt sich u.a. in qualitativen Merkmalen wie Strukturierung, Differenziertheit, (fach-)sprachlicher Gestaltung und Schlüssigkeit der Argumentation.

Im Fach Praktische Informatik besteht die Prüfungsleistung aus der Bearbeitung eines Vorschlags, wofür insgesamt maximal 100 BE vergeben werden können. Ein Prüfungsergebnis von **5 Punkten (ausreichend)** setzt voraus, dass mindestens 45% der zu vergebenden BE erreicht werden. Ein Prüfungsergebnis von **11 Punkten (gut)** setzt voraus, dass mindestens 75% der zu vergebenden BE erreicht werden.

Gewichtung der Aufgaben und Zuordnung der Bewertungseinheiten zu den Anforderungsbereichen

Aufgabe	**Bewertungseinheiten in den Anforderungsbereichen**			**Summe**
	AFB I	**AFB II**	**AFB III**	
1	17	27	16	**60**
2	13	15	12	**40**
Summe	**30**	**42**	**28**	**100**

Die auf die Anforderungsbereiche verteilten Bewertungseinheiten innerhalb der Aufgaben sind als Richtwerte zu verstehen.